SpaceWire-D Prototype and Demonstration System

Networks & Protocols, Long Paper

David Gibson, Steve Parkes
Space Technology Centre
University of Dundee
Dundee, UK
d.z.gibson@dundee.ac.uk

Chris McClements, Stuart Mills
STAR-Dundee Ltd
Dundee, UK

***Abstract*—SpaceWire-D is an extension to the SpaceWire protocol that provides deterministic capabilities over existing SpaceWire equipment. The network is divided into segments using a virtual bus abstraction, where a virtual bus consists of a single RMAP initiator, one or more RMAP targets and the SpaceWire links that make up the paths between the initiator and the targets. Time-codes are broadcast periodically to provide time-division multiplexing, and a network schedule is defined by the allocation of virtual buses to time-slots. If a virtual bus has been allocated a time-slot, it is allowed to execute transactions to any of the targets within the virtual bus as long as the transactions complete their execution before the end of the time-slot. If the schedule is designed so that no virtual buses sharing a link are allocated the same time-slot, packets are no longer affected by blocking which allows the transaction execution times to be calculated and real-time constraints to be satisfied.**

The SpaceWire-D demonstration system has been designed to facilitate the verification of the draft standard. It consists of two RMAP initiators, twelve RMAP targets, a network manager device, a host PC and a routed SpaceWire network to connect the devices together. The LEON2-FT based initiator boards each contain an embedded SpaceWire-D software layer and an automated test scripting system, built on top of the RTEMS real-time operating system. The target boards respond to RMAP commands and provide event notification functionality on the backplane to allow for network activity monitoring. The network manager receives statistics and error information at the end of each schedule epoch, reported by the initiators, and informs the host PC so that it can be read, parsed and displayed to the user. Finally, the host PC runs a suite of software programs to configure, control and monitor the other devices in the demonstration system.

This paper provides an overview of the SpaceWire-D protocol and describes the design and features of the SpaceWire-D demonstration system.

***Index Terms*— SpaceWire, SpaceWire-D, Deterministic Networks, Demonstration System**

I. Introduction

SpaceWire is a data-handling network used on-board spacecraft to provide communication between scientific instruments, mass-memory storage devices, on-board computers, downlink telemetry and other subsystems [1]. SpaceWire enabled devices are connected by full-duplex data links, providing bi-directional data-flow at variable transmission rates of between 2 Mbit/s and 200 Mbit/s. The simplest SpaceWire network can consist of two nodes with a point-to-point link between them. If more complex network topologies are required, routing switches can be used to direct traffic between nodes.

SpaceWire networks can suffer from blocking caused by wormhole routing if a packet is delayed because of another packet currently using one of the links in the packet's path from its source to its destination. The packet will be held within one or more router's buffers until the links are freed and the packet can complete its journey through the network. Due to SpaceWire's arbitrary length packets, this may cause unpredictable packet propagation times which means that a regular SpaceWire network is not suitable for real-time applications such as command and control traffic because these delays could cause a critical deadline to be violated.

The aim of SpaceWire-D is to solve this problem by providing deterministic features in order to ensure that blocking does not cause deadlines to be missed, as well as allowing deterministic and non-deterministic traffic to share the same network. If these goals can be achieved, then cable mass will be reduced as the spacecraft now only requires one network to handle both payload data and control traffic which in turn will reduce complexity and cost.

II. SpaceWire-D

SpaceWire-D is a deterministic extension to SpaceWire designed by the Space Technology Centre at the University of Dundee for ESA [2].

SpaceWire-D operates by controlling which parts of the network are allowed to operate at specific times. Network time is divided into isochronous time-slots which are controlled by the distribution of consecutive SpaceWire time-codes. The network is divided into segments called virtual buses where all traffic, encapsulated within Remote Memory Access Protocol (RMAP) [3] transactions, is controlled by a single initiator. Each initiator has a schedule which describes which time-slots are allocated to its virtual buses. If some rules are adhered to when creating the schedules, the possibility of blocking can be removed and the deterministic requirements of a command and control network can be satisfied.

A. Time-Slots

A SpaceWire-D time-slot is a period of time that begins when an initiator receives a time-code and ends when the initiator receives the next time-code. SpaceWire time-codes contain a 6-bit time-value so there are 64 time-slots. This is illustrated in Figure 1.

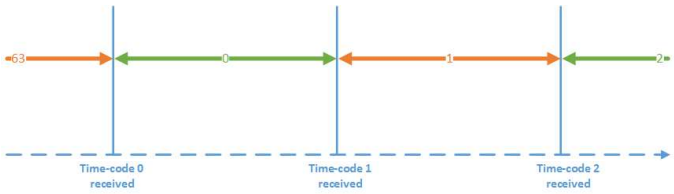


Figure 1: Time-Slots

In Figure 1, there is a timeline going left to right on the horizontal axis showing when time-codes are received by an initiator. At the start of the illustration, time-slot 63 is currently active. When time-code 0 is received by the initiator, this terminates time-slot 63 and signals the beginning of time-slot 0. The same process is repeated for another two time-codes.

The generation of time-codes is synchronised by using a single time-code master responsible for sending out time-codes at fixed-length intervals, typically at a rate of 1-1024 Hz, allowing for between 1 and 16 schedule epochs per second. Each initiator listens for time-codes being received by, for example, installing an interrupt service routine (ISR) that is called whenever a time-code interrupt is raised, or polling a time-code status flag if interrupts are discouraged. The initiator can then inform its SpaceWire-D layer that a new time-slot should be executed, which will in turn execute any scheduled transactions for the virtual bus allocated to the time-slot.

B. Virtual Buses

Virtual buses are segments of the overall network that have a specific structure. They consist of a single RMAP initiator, one or more RMAP targets and the SpaceWire links that make up the paths between the initiator and the targets. For example, take the network architecture illustrated in Figure 2.

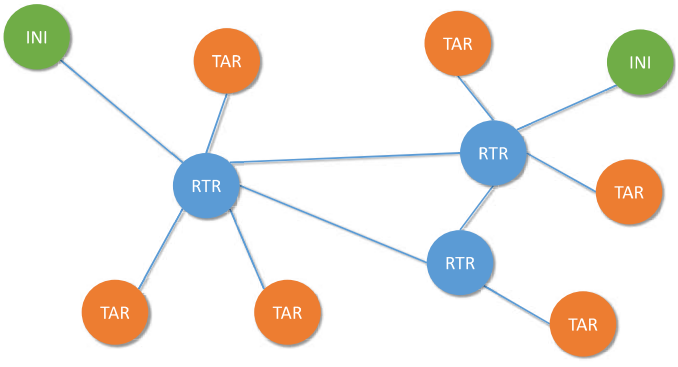


Figure 2: Overall Network Architecture

In Figure 2, there is a network containing two initiators, six targets, three routers and some links to connect the different nodes and routers. Two possible virtual bus configurations are shown in Figure 3.

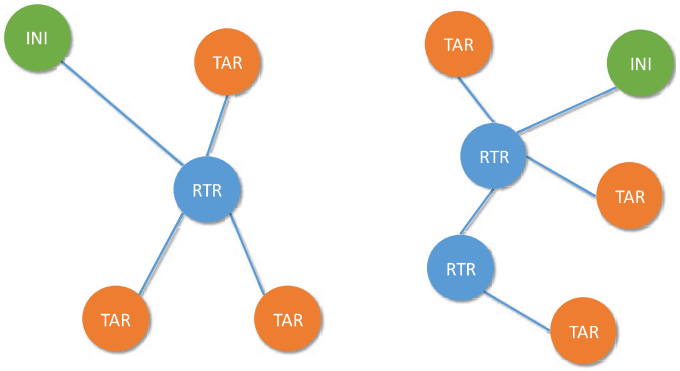


Figure 3: Example Virtual Buses

As shown in Figure 3, there are two virtual buses, each consisting of one initiator, three targets and the links between the nodes. In this example, the two virtual buses have no shared links so they can be thought of as independent i.e. they can operate at the same time without RMAP transactions on one virtual bus interfering with transactions on the other.

Virtual buses have four different functions: an initiator opens a bus, defining its configuration and allocating its time-slots; loads it with transactions, transaction groups or packet transfer requests; executes it during an allocated time-slot; and closes it when it's no longer required. There are four different types of virtual bus, each with their own implementations of the load and execute functions which provide features related to different classes of traffic which exist on a data-handling or command and control network.

1) Static Bus

The Static Bus is the simplest type of virtual bus. It is allocated a single time-slot in which it executes a repeating or single-shot transaction group.

2) Dynamic Bus

The Dynamic Bus can be allocated multiple time-slots and loaded with transaction groups. When it is loaded with a transaction group, the group is executed within the next allocated time-slot that occurs.

3) Asynchronous Bus

The Asynchronous Bus can be allocated multiple time-slots and loaded with prioritised transactions. These transactions are held in a queue and in the time-slot preceding one of the allocated time-slots, a subset of transactions is pulled from the head of the queue until no more will fit in the time-slot or the queue is empty. This transaction group is then executed in the allocated time-slot.

4) Packet Bus

The Packet Bus can be allocated multiple time-slots and loaded with requests to transfer a packet between the initiator and a target. The packet transfer operation takes place in three stages: firstly, the initiator checks the status of a packet channel within the target to make sure the target is ready to receive or send a packet; secondly, the packet is transferred in one or more segments via RMAP read or write transactions depending on if

the initiator is receiving or sending a packet; lastly, the initiator executes an EOP transaction with the target to inform it that the packet has been transferred and that the packet channel may be used to transfer another packet.

III. Demonstration System

The SpaceWire-D demonstration system consists of two LEON2-FT based PXI processor boards, acting as the initiators and controlling the execution of all RMAP transactions; three STAR-Dundee PXI RMAP interface boards [4], each containing four individual RMAP targets with separate memory regions, resulting in a total of 12 RMAP targets; one STAR-Dundee PXI RMAP interface board acting as the network manager, used to receive and store statistics and error information reported by the initiators; two STAR-Dundee PXI 8-port SpaceWire routers, providing the network connecting the devices; and one PXI system controller, running Windows 7, acting as the host PC and running a suite of software used to configure, control and monitor the other devices on the network. A photograph of the SpaceWire-D demonstration system is shown in Figure 4.

Figure 4: SpaceWire-D Demonstration System

In Figure 4, the PXI rack contains the following boards, from left to right: initiator 0, initiator 1, router 0, router 1, the network manager, target interface 0, target interface 1 and target interface 2. To the left of initiator 0, partially in shot, is the host PC.

There are 11 SpaceWire 0.5m cables providing the network between the initiators, targets, routers and network manager. The network architecture and logical addressing has been designed so that both initiators can communicate with targets on the same target interface board without sharing links. This allows, for example, Initiator 0 to communicate with two targets in Target Interface 0 and Initiator 1 to communicate with the other two targets within the same time-slot, without violating the rules of SpaceWire-D. A network architecture diagram for the SpaceWire-D demonstration system is shown in Figure 5.

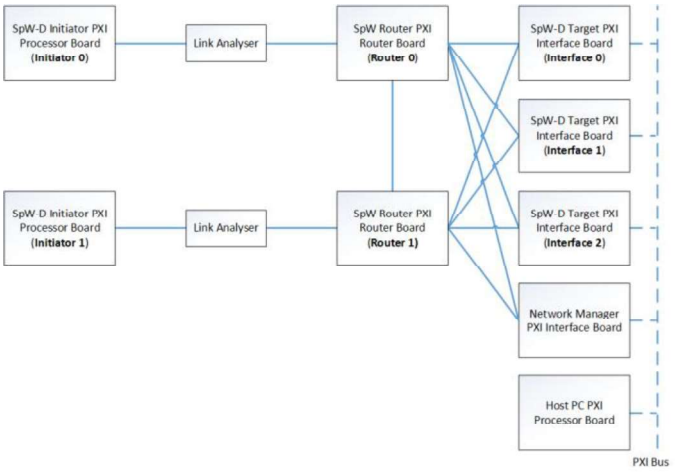


Figure 5: Network Architecture

In Figure 5, the network architecture diagram shows that initiator 0 is connected to router 0 and initiator 1 is connected to router 1. If initiator 0 wants to send an RMAP command to a target, the command is routed from router 0 to SpaceWire port 1 of the relevant target interface board and if initiator 1 wants to do the same, the command is routed from router 1 to SpaceWire port 2 of the target interface board. Commands sent to the network manager from the initiators are routed in a similar manner.

Each of the target interface boards contains four individual RMAP targets with their own logical address and region of memory. Targets 0-3, 4-7 and 8-11 are contained within interface 0, interface 1 and interface 2, respectively. The network manager uses two of its targets; the first is allocated to receive initiator 0's statistics and error reports and the second is allocated to receive reports from initiator 1.

The SpaceWire-D demonstration system uses logical addressing throughout the network to route packets between nodes. The logical addresses and the available memory regions of each device in the network are listed in Table 1.

Table 1: Logical Addresses and Memory Regions

Device	LA	Memory (Start)	Memory (End)
Initiator 0 (I)	0x30	N/A	N/A
Initiator 0 (T)	0x90	0x60000000	0x61000000
Initiator 1 (I)	0x31	N/A	N/A
Initiator 1 (T)	0x91	0x60000000	0x61000000
Target 0	0x40	0x00000000	0x10000000
Target 1	0x41	0x00000000	0x10000000
Target 2	0x42	0x00000000	0x10000000
Target 3	0x43	0x00000000	0x10000000
Target 4	0x50	0x00000000	0x10000000
Target 5	0x51	0x00000000	0x10000000
Target 6	0x52	0x00000000	0x10000000
Target 7	0x53	0x00000000	0x10000000
Target 8	0x60	0x00000000	0x10000000
Target 9	0x61	0x00000000	0x10000000

Target 10	0x62	0x00000000	0x10000000
Target 11	0x63	0x00000000	0x10000000

As listed in Table 1, each node has a logical address and, if the node is a target, a memory region. Each initiator device also contains an RMAP target with a 16 Mbyte region of memory starting at address 0x60000000 and each of the targets within the target interface boards has a 256 Mbyte region of memory starting at address 0x00000000. The target within the initiator devices is used to contain the transaction read and write buffers and to allow the host PC to write data to them before executing a test.

Figure 5 shows that the target interface boards, the network manager and the host PC are connected to the backplane PXI bus. The backplane is used by the host PC to read and write to target memory and receive RMAP command notifications from the targets, as described in Section E.

The interactions between the different devices are illustrated in Figure 6.

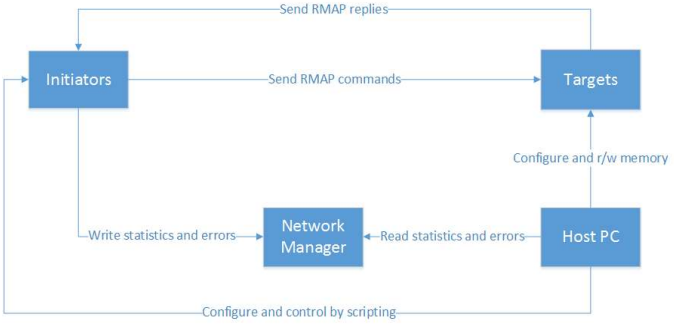


Figure 6: Device Interactions

As shown in Figure 6, each device interacts with one or more other devices in the SpaceWire-D demonstration system. The initiators send RMAP commands to the targets and the targets send RMAP replies back. The initiators report statistics and error information to the network manager, which is then read by the host PC. The host PC configures the initiators using RMAP commands and uploads automated test scripts to control their operation. The targets are configured by the host PC using a combination of RMAP commands and reading/writing to memory on the backplane.

A. Initiators

The initiators are LEON2-FT based PXI processor boards with extensive SpaceWire support. The boards have a SpaceWire router with eight external ports and three internal ports, each connected to independent SpaceWire protocol engines containing three DMA controllers, an RMAP initiator and an RMAP target.

In addition to the embedded SpaceWire-D software layer running on the initiators, which is built on top of the RTEMS real-time operating system [5], there is a demonstrator application. The application is responsible for interpreting scripted commands which are uploaded to the initiators by the Host PC in order to automate test scenarios.

The automated test scripting system allows the user to describe transactions, transaction groups, packet bus operations and time-triggered commands as a text file which is parsed, compiled and uploaded to the initiators by the host PC software. For example, an automated test script could be created that describes 10 transactions encapsulated within 2 transaction groups and a packet bus operation to send a packet from an initiator to a target. The script could then list commands to open two static buses and a packet bus at the start of the test and load them with the transaction groups and packet bus operation at specific times during the execution of the schedule. The automated test scripting system was used to implement all test scenarios during the SpaceWire-D verification activity.

B. Targets

The targets are STAR-Dundee PXI RMAP interface boards which contain a SpaceWire router with four external ports and four internal ports, each connected to an individual RMAP target. The boards have 1 Gbyte of DDR3 memory which can be divided between the four targets as configured by the user. In the case of the SpaceWire-D demonstration system, the targets are configured so that they each have access to 256 Mbytes of memory.

The target boards have the ability to notify a host application whenever certain events occur such as the execution of an RMAP command or a request for command authorisation. The notifications are sent as data structures contained within SpaceWire packets to STAR-System channel 1 on the backplane and can be received using the STAR-System API [6].

Each RMAP comment notification contains the command header parameters as well as the value of the current time-code value in the target board's router so that the time-slot in which the command was executed can be identified. In the SpaceWire-D demonstration system, this information is extracted from the SpaceWire packets by the host PC's software so that it can be used to record and display the activity between the initiators and targets as described in Section E.

C. Routers

The routers are STAR-Dundee PXI routers [4] containing eight external ports and they provide the network for the SpaceWire-D demonstration system, allowing each initiator to be routed to each interface board without sharing any links.

D. Network Manager

The network manager is another STAR-Dundee PXI RMAP interface board. It is controlled by the host PC software to act as the time-code master for the SpaceWire-D network and it also receives statistics and error information reported by the initiators via RMAP write commands to two of the targets within the board.

Each initiator is assigned a separate RMAP target and memory address to write its statistics and error information into at the end of each schedule epoch. Initiator 0 is assigned address 0x00000000 within target 0 and initiator 1 is assigned the same address within target 1.

The host PC's Network Manager software is used to listen for RMAP event notifications coming from the board, which it

parses and uses to read the statistics and error information from address 0x00000000 in the corresponding target. The information is then read from the target by the software and displayed in the Network Manager program running on the host PC. The statistics include the number of completed transactions, incomplete transactions, RMAP errors and early, late and missing time-code errors. Further error information is provided in the error list which describes the time-slot, the virtual bus related to the error and the class and type of error.

Errors are detected at three stages: firstly, if an RMAP command has incorrect header parameters or an error occurs on the initiator where the command cannot be sent, it is reported as an encoder error; secondly, if an RMAP reply is returned to the initiator with an error or if an error occurs on the initiator where the reply cannot be processed, it is reported as a decoder error; lastly, if an RMAP transaction is outstanding at the end of its allocated time-slot, it is cancelled and reported as an incomplete transaction error. The initiators are responsible for detecting and reporting the errors and the network manager is responsible for receiving the error list and informing the host PC, but no further action is taken. It is the responsibility of a higher-level protocol or the application to handle the errors.

E. Host PC

The host PC is an ADLINK PXI-3950 system controller with an Intel Core2 Duo T7500 2.2 GHz processor and 4 GBytes of 667 MHz DDR2 running Windows 7 32-bit. It is responsible for initialising the other devices within the SpaceWire-D demonstration system and running a suite of Qt4.8 based C++ applications used to configure and control the initiators, targets and network manager; and display network activity reported to the network manager via RMAP commands by the initiators, and across the backplane by the targets.

1) Initiator Configuration

The Initiator Configuration program is used to configure and control each of the LEON2-FT processor boards acting as the initiators. It has the ability to read and write the network and target parameters, used by the initiators to calculate RMAP execution times; create different types of virtual buses and assign them to the initiator's schedule; parse, compile and write automated test scripts to the initiators; and send commands to the initiators to enable and disable the schedule and other features like local-timer synchronisation.

2) Target Configuration

The Target Configuration program is used to configure and control each of the RMAP targets in the three PXI interface boards. It has the ability to read and write the RMAP command authorisation parameters; set the packet channel buffer locations and lengths; write data to, and read data from, the target memory; and enable the target interface board as a babbling node. A screenshot of the Target Configuration program is shown in Figure 7.

Figure 7: Target Configuration Program

In Figure 7, the top section allows the user to select which target they would like to configure. In the middle section, the authorisation parameters can be set to define the valid key range, valid target logical address range, accessible memory region and permitted commands. In the bottom section is a tab layout with three separate tabs. The first tab contains a menu to select a packet channel and fields to set the location and length of the receive and transmit buffers used by the packet bus to transfer packets between an initiator and the selected packet channel. The second and third tabs allow the user to write data to, and read data from, the target's memory. Finally, in the second main tab, the user can enable target interface boards as babbling nodes, which send out randomised RMAP commands on the network.

3) Network Manager

The Network Manager program is used to configure the time-code master and receive and display statistics and error information reported to the network manager by the initiators. It has the ability to set the time-code rate and enable or disable the time-code master; display the statistics reported by the initiator in a table, divided by type and time-slot; and display the error information as a list. A screenshot of the error list is shown in Figure 8.

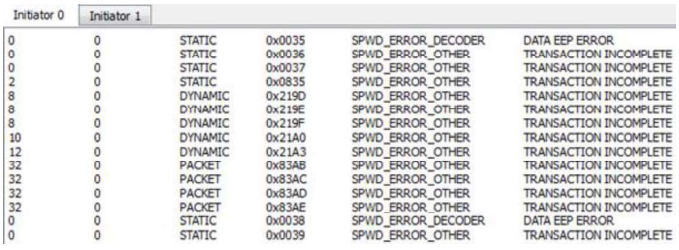


Figure 8: Network Manager Error List

In Figure 8, the screenshot shows a list of errors reported by the initiator during a test in which a STAR-Dundee Link Analyser Mk2 is periodically injecting disconnect errors in the link between router 0 and target interface 0. The columns are:

virtual bus ID, target index, virtual bus type, transaction ID, error category and error type. In this example, two types of errors are detected and reported: firstly, if the disconnect causes the command packet to be truncated, it will not have a corresponding reply so at the end of the allocated time-slot, the transaction is cancelled and reported as an incomplete transaction; and secondly, if the disconnect causes the reply packet to be truncated in its data section, it is reported as a data EEP decoder error.

When the Network Manager program is initialised, it starts to listen for RMAP event notifications from the Network Manager RMAP interface board by receiving SpaceWire packets on the backplane through STAR-System channel 1. When a notification is received, the software checks that the parameters of the RMAP command match those expected by an initiator statistics and error report. If so, the report is read from the target memory and the statistics table for the relevant initiator is updated and any errors detected during the last schedule epoch are added to the initiator's error list.

4) Target Monitor

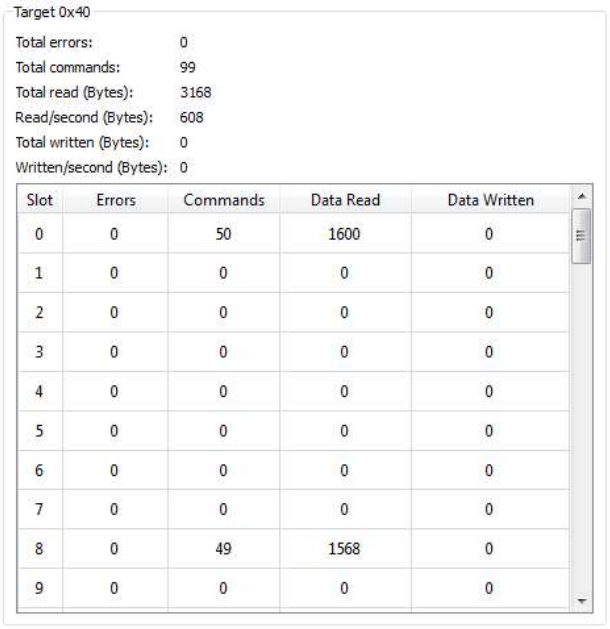
The Target Monitor program is used to display the network activity visually and statistically through a series of views. It has the ability to display activity in real-time, by updating a grid that shows if any of the targets were read from or written to during each time-slot. It shows the number of completed transactions, bytes read from and written to the target in total and per second, and it also breaks this information up for each time-slot. Finally, it shows a list of detailed information about all RMAP transactions taking place across all targets. There are three views in the Target Monitor program: the schedule view, the target statistics view and the command list view.

A screenshot of the Target Monitor schedule view, which shows network activity as a grid, is shown in Figure 9.

Figure 9: Target Monitor Schedule View

In Figure 9, the screenshot shows the Schedule View during the execution of a test where each initiator is executing two static buses and one dynamic, asynchronous and packet bus. The virtual buses in initiator 0 are executing transactions with targets 0x40-0x43 and 0x50-0x51, taking up the left side of the diagram. Initiator 1's virtual buses are executing transactions with targets 0x52-0x53 and 0x60-0x63, shown on the right side of the diagram. There are two static buses executed by each initiator, shown as dark blue cells, and allocated to time-slots 0 and 2. The dynamic bus executed by each initiator, shown as green cells, are allocated time-slots 8, 10 and 12. The asynchronous bus executed by each initiator, shown as magenta cells, is allocated time-slot 16. Finally, the packet bus executed by each initiator, shown as cyan cells, is allocated time-slots 32, 34 and 36.

The target statistics view lists the number of errors, commands and bytes read/written in total, per second and divided by time-slot. Figure 10 shows an image of the Target Statistics View section for target 0x40.



Slot	Errors	Commands	Data Read	Data Written
0	0	50	1600	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	49	1568	0
9	0	0	0	0

Figure 10: Target Monitor Target Statistics View

In Figure 10, the screenshot shows the Target Statistics View for target 0x40 during the execution of a schedule containing network activity in time-slots 0 and 8. The total and per second statistics are shown in the top section and the per time-slot statistics are shown in the scrollable table.

The final section of the Target Monitor program is the Command List View, which displays a detailed description of every RMAP command received on all targets. An image of the Command List View is shown in Figure 11.

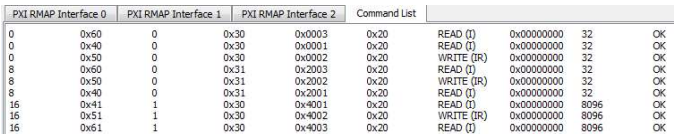


Figure 11: Target Monitor Command List View

In Figure 11, the screenshot shows the start of the Command List View during the execution of a schedule containing at least three static buses. The columns are: virtual bus ID, target logical address, target index, initiator logical address, transaction ID, RMAP key, command type, memory address, data length and status. In this case, there are nine transactions executed by static buses 0, 8 and 16. The first three, to targets

0x40, 0x50 and 0x60, and the last three, to targets 0x41, 0x51 and 0x61 are executed by initiator 0x30. The middle three, to targets 0x40, 0x50 and 0x60, are executed by static bus 8 in initiator 0x31.

IV. Conclusions

SpaceWire-D is an extension to the SpaceWire protocol that provides deterministic capabilities over existing equipment. It does this by using time-division multiplexing and a virtual bus system to schedule traffic on the network so that no blocking can occur, resulting in reliable RMAP transaction execution times.

The SpaceWire-D demonstration system was designed to verify the SpaceWire-D standard and demonstrate its capabilities. It consists of a PXI rack containing two initiators, twelve targets, a network manager, a host PC and a routed SpaceWire network to connect the devices together. An embedded SpaceWire-D layer and automated test scripting system was designed, built on top of the RTEMS real-time operating system; and a software suite, running on the host PC, was designed to configure, control and monitor the other devices on the network.

Acknowledgements

The research leading to these results has received funding from the European Space Agency under ESA contact number 4000107346/12/NL/LvH/fe. We would also like to thank David Jameux, the ESA project manager for the SpaceWire-D related activity.

References

[1] ECSS Standard ECSS-E-ST-50-12C, "SpaceWire – Links, nodes, routers and networks", Issue 2, European Cooperation for Space Standardization, 31 July 2008, available from http://www.ecss.nl

[2] S. Parkes, D. Gibson, A. Ferrer Florit, "SpaceWire-D Standard Draft E", Issue 0.4, Space Technology Centre, University of Dundee, April 2015

[3] ECSS Standard ECSS-E-ST-50-52C, "SpaceWire – Remote memory access protocol", Issue 1, European Cooperation for Space Standardization, 5 February 2010, available from http://www.ecss.nl

[4] STAR-Dundee, "SpaceWire PXI", https://www.star-dundee.com/products/spacewire-pxi

[5] The RTEMS Project, "RTEMS Real Time Operating System (RTOS)", https://www.rtems.org/

[6] S. Mills, S. Parkes, "A Software Suite for Testing SpaceWire Devices and Networks", DASIA International Space System Engineering Conference, Barcelona, Spain, August 2016